Continuous Integration Report

"Mathochist Studios" Cohort 4, Team 11

Euan Cottam
Charlie Thoo-Tinsley
Harri Thorman
Will King
Zach Moussallati
Aiden Turner
Marcus Williamson
Joshua Zacek

Methods & Approaches

For the project's development, our implementation team was relatively small; however, to maintain code quality, it was pivotal to review each other's code. Since most of the development was done remotely, fast feedback from peer reviews was easier to obtain as each contributor could access the code remotely from our GitHub page. With automated testing that continuous integration practices provides, it gave us the opportunity to gather quick feedback at every commit to the repository & meant that when it came to reviewing each other's code, the practices that came with CI made it much easier.

In this project, as discussed in our Method Selection & Planning documentation, we followed an agile methodology. In practice, this led to development that involved more minor, more frequent commits to a single main branch, rather than larger, infrequent commits to separate branches & then later merging them. This motivated us to adopt trunk-based development, as it naturally fit into our existing workflows. This trunk-based approach meant that automated testing, along with manual testing, could be performed more easily than would have been possible if we had not followed it. It also meant that other team members could review code manually, since there was less code to review. This allowed us to maintain high-quality code, as members could provide better feedback on code committed because they didn't have to review as much code. However we did allow flexibility in our method with some branches being created for sections like if two people was working on a popular class like main.java for example as clashes could frequently occur so it was easier to ensure they would be the minimum amount of clashes required.

For automated tests, each commit would run a test to ensure the project could be built on all three operating systems specified in the specification: macOS, Linux & Windows. This allowed us to catch errors early, preventing them from snowballing and increasing the workflow later on. It also helped during manual unit testing, as the testing team was always able to build & test on the most up-to-date version of the application, increasing the validity of their tests, if they were doing manual tests. However, this was less of a concern, as unit tests would run automatically each time a commit was made, providing fast feedback & meant manual testing wasn't needed.

Continuous Integration Infrastructure

We have been using continuous integration to automate and check the work and implementation of the game so that we spend less time checking and testing and can instead spend time doing coding and adding more features to the game. It also helps to ensure that our project compiles and runs on every platform without having to individually open and test each platform which would take a long time.

Our current GitHub actions work by trigger once there has been either a push to the main branch, a pull from the main branch to another branch or a scheduled time which is currently set to being 5pm on Monday. This ensures that any changes made to the main branch which is used for building and releases, is always checked for common problems and ensures that it is working for any releases made. Our actions are split into two sections, the dependency submission which compiles the game and creates and submits a dependency graph to the artifacts so it's viewable afterwards for testing and debugging purposes to tell what happened or went wrong in the event it goes wrong.

We then built the game on each operating system we planned to support, this being macOS, windows and Ubuntu Linux. It is specified to be the latest version that GitHub supports of each OS to ensure it always works on the devices people are most likely to use. Any errors in build will cause the action to fail and a release will not be made as to not submit a broken release to main which people could potentially download and be confused as to why it does not work. Once we have ensured each operating system is working and can be built, we automatically run the test we have created previously to ensure all game content works correctly using the build on ubuntu we just made as an example.

If all tests work and the game is correct then we can push the build on ubuntu to being a release if we are being run to due to being at 5pm on Monday. This prevents too many releases being made for small changes like bug fixes and will fill up our GitHub actions very fast causing further releases to become impossible which will be counterintuitive. Therefore, we will only store builds and set them as releases when we reach time required.

Finally, if any files relating to failures have been made it will be uploaded to artifacts for review and to check if anything went wrong when doing the work. For the final build we upload, we may disable the automatic releases so that our GitHub actions storage is not filled up and used on a project we may have stopped supporting a long time ago.